CS 144, Winter 2024: Project 1 ("p1bomb"): Defusing a Binary "Bomb"
Due: Tuesday, Jan 23 at 11:59pm

This project involves "defusing" a "binary bomb". The bomb is an executable binary, compiled from C code, consisting of a sequence of **six phases**, each of which prompts you to **enter a string**. If you type a correct string, then the phase is ***defused*** and the bomb proceeds to the next phase. Otherwise, the bomb terminates. The bomb is defused when every phase has been defused. Your goal is to defuse your bomb before the due date.

Step 1: Get Your Bomb

To obtain your bomb, use a web browser to go to the bomb server http://bomb.cs.uchicago.edu:15213. However, this server is only accessible within uchicago network, so there are few ways to get there:
a) You are physically on campus and connect to the wifi network
b) Install cVPN
c) Use vDesk (Remote Desktop Linux) https://howto.cs.uchicago.edu/techstaff:vdesk?s[]=vdesk

After you figure out a way to access http://bomb.cs.uchicago.edu:15213, you need to fill the form in order to get a bomb that is uniquely generated for you. On the first line of the form, you should enter your CNetID. On the second line of the form, enter your email address, then click "Submit". The server will build your customized bomb and return it to your browser in a `.tar` file named `bombN.tar`, where N is the unique number of *your* bomb. If you make a mistake requesting a bomb (such as neglecting to save it), request another one. If you are frustrated by having gotten too many explosions (say, before you learn how to set breakpoints), and want to start again, you can also just restart with a new bomb. However, every bomb generated by the server is different, and requires different solutions.

Step 2: Run Your Bomb

Running your bomb is the first thing to do before you can actually start to defuse it, so read these instructions carefully. The bomb that you just downloaded **will only run on one of** these machines below (which you can `ssh` into), all ending with `.cs.uchicago.edu`:

VPN is **not** required at: `linux1` `linux2` `linux3` `linux4` `linux5` `linux6` `linux7`

VPN is **required** at these machines below:

machoke	machamp	bellsprout	weepinbell	victreebel	tentacool
tentacruel	geodude	graveler	golem	ponyta	rapidash
slowpoke	slowbro	magnemite	magneton	farfetchd	doduo
seel	dewgong	grimer	muk	shellder	cloyster
gastly	haunter	gengar	onix	drowzee	hypno
krabby	kingler	voltorb	electrode	exeggutor	cubone
marowak	hitmonchan	lickitung	koffing		

Pick any of that server and make sure you can ssh into it (i.e. `ssh` **`CNET`**`@hitmonchan.cs.uchicago.edu`). If you feel that your server is slower (or unresponsive), please use different one. Once you're in, you will be able to find your **`CNET`**`-cs144-win-24/p1bomb` directory. Now, you must copy your `bombN.tar` file into this directory, `svn add` it, and commit it. You can use `scp` to upload your bomb to this directory. At your Mac/Windows terminal, enter into your local directory that stores the bomb and run a command like

`scp bomb`**`N`**`.tar` **`CNET`**`@linux.cs.uchicago.edu:/home/`**`CNET`**`/cs144/`**`CNET`**`-cs144-win-24/p1bomb`

This is just an example. You can put your bomb in any directory you want, but if you are not sure, just follow the instruction above.

Also, please read this for more informations about `scp`.

Then `tar xvf bombN.tar`. This creates a `./bombN` directory containing:

- `README`: Identifies the bomb and its owner.
- `bomb`: The executable binary bomb.
- `bomb.c`: Source file with the bomb's `main()` routine.

Finally, to run your bomb:
`cd bomb`**`XX`**
`gdb bomb`
`b` `phase_1`
`r`
(Note: `b` is for putting a breakpoint, and `r` is for starting the program execution from the beginning of the program. More about GDB commands is here http://www.yolinux.com/TUTORIALS/GDB-Commands.html)

Step 3: Defuse Your Bomb

Reading through `bomb.c` will show you the basic steps the bomb uses to process the string inputs through its six phases. You do not get to see the source for the `phase_1`, `phase_2`, ... functions (called from `main()`). You have to reverse-engineer them. Your job is to "defuse" the bomb, which simply means that the bomb execution has to reach its final `return 0`. Defusing each phase earns you 10 points; an ideal score is 60 points.

You can use many tools to help you with this; please look at the **Hints** section below for some tips and ideas. The best way is to use a debugger to step through the disassembled binary.

If your bomb "explodes" it reports to the bomb scoreboard at http://bomb.cs.uchicago.edu:15213/scoreboard. **Be careful!** With each explosion you will lose 1/2 point (up to a max of 20 points) in the final score for the project. We do round up to an integral score, so the first explosion is "free". Every bomb is different, so the required solutions are also different.

The phases get progressively harder to defuse, but expertise you gain as you move from phase to phase should offset this difficulty. Even more so than with the previous project, there is no way to do this project in one night: it takes time, care, and insight. **Start now.**

The bomb ignores blank input lines. If you run your bomb with a command line argument, for example,

```
linux>  ./bomb sol.txt
```

then it will read the input lines from `sol.txt` until it reaches EOF (end of file), and then switch over to `stdin`. This feature saves you from having to repeatedly retype the solutions to phases you have already defused.

To avoid accidentally detonating the bomb, you will need to learn how to single-step through the assembly code and how to set breakpoints (this is covered in Lab 2). You will also need to learn how to inspect both the registers and the memory state. A nice side-effect of doing the project is that you will get very good at using a debugger.

Handin

There is nothing to hand in. The bomb will report to the scoreboard when a phase is defused. You can track how you (and others) are doing at: http://bomb.cs.uchicago.edu:15213/scoreboard. Remember, **bomb.cs.uchicago.edu** is only accessible by using VPN or vDesk. This web page is updated every 30 seconds to show the progress on each bomb. You know which line corresponds to you because you know the number N of your bomb `bombN`.

Hints

Basically, there are many ways to defuse your bomb. You can examine it in great detail without ever running it, and figure out exactly what it does. This is a useful technique, but not always easy to do. **You can also run it under a debugger, watch what it does step by step, and use this information to defuse it. This is probably the *fastest way*.**

A bad strategy would be brute force: writing a program that tries every possible key to find the right one. This will not work, because you lose credit with every explosion, and there are too many strings to try.

There are many tools which are designed to help you figure out both how programs work, and what is wrong when they do not work. Here are some tools that may be useful:

- `gdb` The GNU debugger, the command line debugger tool available on the CSIL machines, and covered in Lab 2. You can trace through a program line by line, examine memory and registers, look at both the source code and assembly code (we are not giving you the source code for most of your bomb), set breakpoints, set memory watch points, and write scripts. Here are some tips for using `gdb`.
  - To keep the bomb from blowing up every time you type in a wrong input, **set breakpoints**. Being careless about this is the easiest way to lose points. (I personally will put "`break explode_bomb`") everytime I am running the gdb.
  - The CS:APP Student Site at http://csapp.cs.cmu.edu/public/students.html has a single-page `gdb` summary.
  - For other documentation, type "`help`" at the `gdb` command prompt, or type "`man gdb`", or "`info gdb`" at a Unix prompt. Some people also like to run `gdb` under `gdb-mode` in `emacs`.
  - If you want, you can see the disassambly result and register at the same place. You need to put these lines in "`$HOME/.gdbinit`". Create that file if does not exist.

    ```
    layout regs
    tui reg all
    ```

- `objdump -t` This command will print out the bomb's symbol table. The symbol table includes the names of all functions and global variables in the bomb, the names of all the functions the bomb calls, and their addresses. You may learn something by looking at the function names!
- `objdump -d` Use this to disassemble all of the code in the bomb. You can also just look at individual functions. Reading the assembler code can tell you how the bomb works.

  Although `objdump -d` gives you a lot of information, it does not tell you the whole story. Calls to system-level functions are displayed in a cryptic form. For example, a call to `sscanf` might appear as:

  ```
  8048c36:  e8 99 fc ff ff  call   80488d4 <_init+0x1a0>
  ```

To determine that the call was to `sscanf`, you would need to disassemble within `gdb`.

- `strings`

  This utility will display the printable strings in your bomb, but the results will not be very informative (though possibly amusing). Your bomb expects you to supply certain strings, but the validity of the string is not assessed by simply comparing it to some internally stored correct string. Rather, the strings you provide to your bomb have to be crafted according to your understanding of the bomb's computation.

Acknowledgments

This is based on the "bomblab" project developed by the authors of our textbook.